Magda's notes about **compression**

Last update: July 2, 2021

Informal notes for my future self who is likely to forget. I explain the papers the way I understand them, using terminology and logic natural to me. This means I may deviate from the original paper structure, notation, etc. At places, my interpretation my be incorrect due to lack of understanding. I will strive for this not to happen too often but I'm certainly not infallible.

This is a working document, not polished, with possible typos, editing errors, etc.

Contents

1 Intro thougths

maximum likelihood we assume a population $\mathbf{x} \in \mathbb{X}$ governed by some unknown probablity distribution $p(\mathbf{x})$. We have at our disposal a sample from this population $\mathbb{S} = \{\boldsymbol{x}_i\}_{i=1}^n$ (assume i.i.d.) and we use it to infer a statistical model of the data. In maximum liekelihood approach we posit a distribution family $p(\mathbf{x}|\boldsymbol{\theta})$ and we search for the best parameters $\boldsymbol{\theta}^*$ so that the distribution $p(\mathbf{x}|\boldsymbol{\theta}^*)$ best fits the observed data $\sim$ we search parameters that would have most likely produced the data.

The max *likelihood* problem is thus $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \prod_i^n p(\boldsymbol{x}_i|\boldsymbol{\theta})$ which is equivalent to minimizing the negative log of the likelihood

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} - \sum_i^n \log p(\boldsymbol{x}_i|\boldsymbol{\theta}), \quad \boldsymbol{x}_i \sim p(\mathbf{x}) \ . \tag{1.1}$$

compression the theoretical lower bound (thanks to Shanon) on the expected length of a binary code for random variable $\mathbf{x}$ is the entropy of the distribution $H(p) = -E_{p(\mathbf{x})} \log_2 p(\mathbf{x})$, where each sample $\boldsymbol{x}$ is encoded with the shortest possible binary code of length equal to its information content $h(\boldsymbol{x}) = -\log_2 p(\boldsymbol{x})$.

Since the true data distribution $p(\mathbf{x})$ is unknown, we cannot use it for compression. We use instead a compression model, a distribution $p(\mathbf{x}|\boldsymbol{\theta})$, to establish the lengths of the codewords for the symbols $\boldsymbol{x}$ as $l(\mathbf{x}) = -\log_2 p(\boldsymbol{x}|\boldsymbol{\theta})$. These lengths are clearly not the same as the information content of the individual observations $\boldsymbol{x}$. The expected codeword length for a randomly sampled symbol $\mathbf{x}$ is the cross-entropy $CH(p,q) = -E_{p(\mathbf{x})} \log_2 p(\mathbf{x}|\boldsymbol{\theta})$.

To minimize the expected codeword length of symbols sampled from the unknown $p(\mathbf{x})$, we shall minimize the cross-entropy $\boldsymbol{\theta}^* = \arg\min_\theta -E_{p(\mathbf{x})} \log_2 p(\mathbf{x}|\boldsymbol{\theta})$. In practice, this can be achieved by minimizing the empirical estimate over the data sample

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} - \frac{1}{n} \sum_i^n \log_2 p(\boldsymbol{x}_i|\boldsymbol{\theta}), \quad \boldsymbol{x}_i \sim p(\mathbf{x}) \ . \tag{1.2}$$

Note that this is equialent to problem (1.1).

> Maximizing likelihood is equivalent to finding a compression model that will minimize the expected codeword length.

Furthermore we can develop the cross-entropy as follows

$$\begin{aligned}
CH(p,q) &= -E_{p(\mathbf{x})} \log_2 p(\mathbf{x}|\boldsymbol{\theta}) \\
&= E_{p(\mathbf{x})} \log_2 \frac{p(\mathbf{x})}{p(\mathbf{x}|\boldsymbol{\theta})p(\mathbf{x})} \\
&= E_{p(\mathbf{x})} \log_2 \frac{p(\mathbf{x})}{p(\mathbf{x}|\boldsymbol{\theta})} - E_{p(\mathbf{x})} \log_2 p(\mathbf{x}) \\
&= D_{\mathrm{KL}}(p,q) + H(p) \ . \qquad (1.3)
\end{aligned}$$

Since the unkown p and hence the entropy $H(p)$ are fixed, minimizing the cross entropy $H(p,q)$ is also equivalent to minimizing the KL divergence between p and q which can be interpreted as

- minimizing the statistical distance between the distribution (bringing q close to p)
- minimizing the additional bits from using q instead of p to establish the codewords.

2 Freay and Hinton: Bits-back with arithmetic coding

Paper: Brendan J. Frey and Geoffrey E. Hinton. “Efficient Stochastic Source Coding and an Application to a Bayesian Network Source Model”. In: *Computer Journal* 40.2/3 (1997), pp. 157–165

Notes taken: 4/6/2021

TLDR: In a mixture model it may be difficult to directly evaluate $p(\mathbf{x})$ so that you can transmit $\mathbf{x}$ by entropy encoder. You may hence want to use the prior $p(\mathbf{y})$ and the conditionals $p(\mathbf{x}|\mathbf{y})$ to transmit the message $(\mathbf{y}, \mathbf{x})$. However, for each $\boldsymbol{x}$, there are multiple different codewords with different length $l(\boldsymbol{y}, \boldsymbol{x})$ depending on the choice of $\boldsymbol{y}$ and hence the conditional $p(\boldsymbol{x}|\boldsymbol{y})$. A natural choice is the shortest such codeword. They show and demonstrate on experiments, that you can do better (and theoreticaly get to the optimal expected transmission length $-\log_2 p(\boldsymbol{x})$) in a bits-back coding scheme in which the encoder uses some auxiliary data to sample $\boldsymbol{y}$ that will be transmitted determining also the corresponding $p(\boldsymbol{x}|\boldsymbol{y})$ used for encoding $\boldsymbol{x}$. Since the decoder can recover these auxiliary data it gets some bits for free without them being directly included in the transmission. These are the bits back that reduce the effective transmission length.

2.1 Intro

A *source code* uses as *source model* to map each input symbol to a unique *codeword*. There are two principal source models: **one-to-one** where each symbol is mapped to a single codeword; **one-to-many** where each symbol is mapped to a distribution across multiple codewords. One-to-many source models arrise naturally in many domains such as in mixture distributions

$$\mathbf{x} \in \mathbb{X} \sim p(\mathbf{x}) = \sum_y p(\mathbf{y}) p(\mathbf{x}|\mathbf{y}) \ . \tag{2.1}$$

.

For example for the mixture of Gaussian in figure 1 the optimal one-to-one model uses codewords with length given by the information content of the symbols $l(\mathbf{x}) = h(\mathbf{x}) = -\log_2 p(\mathbf{x})$.

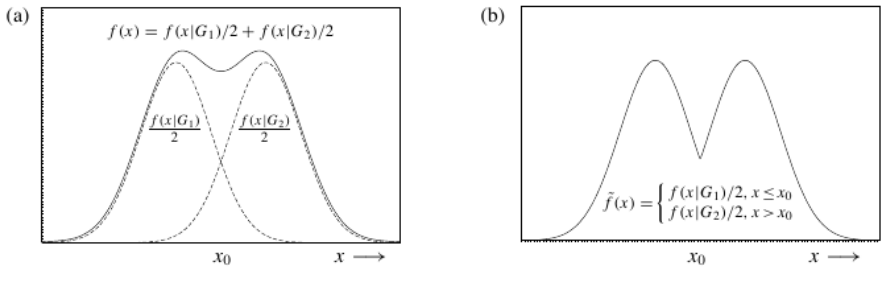

Figure 1: Mixture of two Guassians.

In one-to-many model, we have for each symbol $\boldsymbol{x}$ two possible codewords: one given by G_1 the other by G_2 with the lengths given by $h_{\boldsymbol{x}|\boldsymbol{y}}(\boldsymbol{x}) = -\log_2 p(\mathbf{x}|\mathbf{y} = \mathrm{y}), \mathrm{y} \in \{1, 2\}$ respecitvely. Assuming the prior $p(\mathbf{y} = 1) = p(\mathbf{y} = 2) = 0.5$ we also have $h_{\boldsymbol{y}}(1) = h_{\boldsymbol{y}}(2) = -\log_2 0.5 = 1$ bit to communicate $\mathbf{y}$ and hence the identity of the conditional distribution.

A natural choice is to always pick the shorter of the two codewords for $\boldsymbol{x}$. This corresponds to a distribution in (b) in figure 1 which is clearly suboptimal (gives longer codes then the one-to-one) around $\boldsymbol{x}_0$.

Here they show how stochastic one-to-many model relying on the *bits-back* idea can achieve same rates as the one-to-one scheme.

Running example: We want to encode a symbol $\boldsymbol{x}$ that is twice as likely under G_1 then G_2 so that it requires 2 and 3 bits respectively under the two distributions (that is $p(\boldsymbol{x}|\mathbf{y}=1)=1/4$ and $p(\boldsymbol{x}|\mathbf{y}=2)=1/8$). With 1 bit to specify the $\mathbf{y}$ and hence the conditional distribution we get the total lenght of 3 and 4 bits for the two-level code. We now have the following options:

1. picking alwyas the shorter code gives expected length of 3 bits;
2. picking between the two codes equally often (according to the prior $p(\mathbf{y})$) gives expected length of 3.5 bits;
3. use bits-back to get the expected length 2.5 bits.

This is the bits-back idea: Instead of picking always the shortest codeoword or pick equally often amongst the codeword in the one-to-many setup, you will be picking $\boldsymbol{y} \in 1, 2$ randomly according to some probability $q(\mathbf{y}|\mathbf{x})$. However, to select (sample) $\mathbf{y}$ you will use some random sampler based on transorming some auxiliary data $\mathbf{z}$ into the samles $\mathbf{y} \sim q(\mathbf{y})$. A trivial example for univariate case is the inverse CDF sampling starting from uniform $\mathbf{z}$ or, as they suggest here, the arithmetic decoding which can produce vectorial $\mathbf{y}$ starting from a random binary sequence $\mathbf{z}$. Once you sampled $\boldsymbol{y}$ you will encode it using $p(\mathbf{y})$ and concatenate with the encoding of $\boldsymbol{x}$ using the corresponding $p(\boldsymbol{x}|\boldsymbol{y})$. The message you transmit thus has length $l(\boldsymbol{x},\boldsymbol{y}) = -\log_2 p(\boldsymbol{x}|\boldsymbol{y}) - \log_2 p(\boldsymbol{y})$.
The reciever has access to the encoding models $p(\mathbf{y}), p(\mathbf{x}|\mathbf{y})$ and it can thus easily decode the symbols $\mathbf{x}$ and $\mathbf{y}$ from the codewords $C(\mathbf{x},\mathbf{y})$. However, it also has access to the sampling distribution $q(\mathbf{y})$ and the algirithm the encoder uses to sample $\mathbf{y}$ by transforming $\mathbf{z}$. It can thus reverse this operation (so the transformation must be one-to-one) and recover $\mathbf{z}$ at no additional communication cost - these are the 'bits-back'. If $\mathbf{z}$ contains some useful information, these 'bits-back' essentially reduce the *effective message length*.

In our **running example** lets imagine $\mathbf{z} \in a, b$ with $p(\mathbf{z}=a) = p(\mathbf{z}=b) = 0.5$. Then once we recover $\mathbf{z}$ we have communicated $h_{\mathbf{z}}(\boldsymbol{z}) = -\log_2 0.5 = 1$ bit of information at no cost and so the **expected effective length is** $3.5 - 1 = 2.5$ **bits**.

How many bits can be recovered is a function of the probability distribution $q(\mathbf{y}|\mathbf{x})$.

The **average code length** for a symol $\boldsymbol{x}$ is

$$\mathcal{E}(\boldsymbol{x}) = \sum_y q(\boldsymbol{y}|\boldsymbol{x}) l(\boldsymbol{x},\boldsymbol{y}) \ . \tag{2.2}$$

Here $l(\boldsymbol{x},\boldsymbol{y}) = -\log_2 p(\boldsymbol{x}|\boldsymbol{y}) - \log_2 p(\boldsymbol{y})$ are the different lengths using different conditionals to encode $\boldsymbol{x}$ depending on the choice of $\boldsymbol{y}$.

The average information content (the entropy) of the selecting distribution

$$\mathcal{H}(\boldsymbol{x}) = -\sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 q(\boldsymbol{y}|\boldsymbol{x}) \tag{2.3}$$

are **the average bits back we can recover** for this symbol $\mathbf{x}$ when choosing $\boldsymbol{y}$ according to $q(\mathbf{y}|\boldsymbol{x})$.

For this you need to recall that $q(\boldsymbol{y}|\boldsymbol{x})$ is the coding model which is used to get from the auxiliary $\boldsymbol{z}$ to $\boldsymbol{y}$ - $\boldsymbol{z}$ is the codeword for y under the coding model $q(\boldsymbol{y}|\boldsymbol{x})$ with length $l(\boldsymbol{y}) = -\log q(\boldsymbol{y}|\boldsymbol{x})$ are these are the bits we get back. So the above example of using inverse CDF is probably nonsense.

The average effective codeword length of symbol $\boldsymbol{x}$ is the difference between the two above

$$\mathcal{F}(\boldsymbol{x}) = \mathcal{E}(\boldsymbol{x}) - \mathcal{H}(\boldsymbol{x}) \ . \tag{2.4}$$

This is also called the ***free energy*** - same concept known from statistical physics.

Next comes the question what $q(\mathbf{y}|\boldsymbol{x})$ shall be. They claim it shall be the Boltzman distribution based on the codeword lengths

$$q^*(\boldsymbol{y}|\boldsymbol{x}) = \frac{2^{-l(\boldsymbol{x},\boldsymbol{y})}}{\sum_{\boldsymbol{y}'} 2^{-l(\boldsymbol{x},\boldsymbol{y}')}} \tag{2.5}$$

My proof for Boltzman:

$$\begin{aligned}\mathcal{F}^*(\boldsymbol{x}) = \mathcal{E}^*(\boldsymbol{x}) - \mathcal{H}^*(\boldsymbol{x}) &= \sum_y q^*(\boldsymbol{y}|\boldsymbol{x}) l(\boldsymbol{x},\boldsymbol{y}) + \sum_y q^*(\boldsymbol{y}|\boldsymbol{x}) \log_2 q^*(\boldsymbol{y}|\boldsymbol{x}) \\ &= \sum_y \frac{2^{-l(\mathbf{x},\mathbf{y})}}{\sum_{y'} 2^{-l(\mathbf{x},\mathbf{y}')}} \left[l(\mathbf{x},\mathbf{y}) + \log_2 \frac{2^{-l(\mathbf{x},\mathbf{y})}}{\sum_{y'} 2^{-l(\mathbf{x},\mathbf{y}')}} \right] = l(\mathbf{x},\mathbf{y}) - l(\mathbf{x},\mathbf{y}) - \log_2 \sum_{y'} 2^{-l(\mathbf{x},\mathbf{y}')} \\ &= -\log_2 \sum_{y'} 2^{-l(\mathbf{x},\mathbf{y}')} = -\log_2 \sum_{y'} 2^{\log_2 p(\boldsymbol{x},\boldsymbol{y})} = -\log_2 \sum_{y'} p(\boldsymbol{x},\boldsymbol{y}) = -\log_2 p(\boldsymbol{x})\end{aligned}$$

The Boltzman brings the free energy to the same lengths as if one-to-one code with $p(\mathbf{x})$ were used so it is optimal. □

I'd think it shall be the posterior $p(\boldsymbol{y}|\boldsymbol{x}) = p(\boldsymbol{y})p(\boldsymbol{x}|\boldsymbol{y})/p(\boldsymbol{x})$.

My proof for posterior:

$$\begin{aligned}\mathcal{F}^*(\boldsymbol{x}) = \mathcal{E}^*(\boldsymbol{x}) - \mathcal{H}^*(\boldsymbol{x}) &= \sum_y p(\boldsymbol{y}|\boldsymbol{x}) l(\boldsymbol{x},\boldsymbol{y}) + \sum_y p(\boldsymbol{y}|\boldsymbol{x}) \log_2 p(\boldsymbol{y}|\boldsymbol{x}) \\ &= \sum_y p(\boldsymbol{y}|\boldsymbol{x}) \left[-\log_2 p(\boldsymbol{x},\boldsymbol{y}) + \log_2 p(\boldsymbol{y}|\boldsymbol{x}) \right] = \sum_y p(\boldsymbol{y}|\boldsymbol{x}) \left[-\log_2 p(\boldsymbol{x}) \right] = -\log_2 p(\boldsymbol{x})\end{aligned}$$

The posterior brings the free energy to the same lengths as if one-to-one code with $p(\mathbf{x})$ were used so it is also optimal. □

Does this mean that the Boltzman and the posterior distributions are the same?

$$\begin{aligned}q^*(\boldsymbol{y}|\boldsymbol{x}) &= \frac{2^{-l(\boldsymbol{x},\boldsymbol{y})}}{\sum_{\boldsymbol{y}'} 2^{-l(\boldsymbol{x},\boldsymbol{y}')}} = \frac{2^{\log_2 p(\boldsymbol{x},\boldsymbol{y})}}{\sum_{\boldsymbol{y}'} 2^{\log_2 p(\boldsymbol{x},\boldsymbol{y}')}} \\ &= \frac{p(\boldsymbol{x},\boldsymbol{y})}{\sum_{\boldsymbol{y}'} p(\boldsymbol{x},\boldsymbol{y}')} = \frac{p(\boldsymbol{x},\boldsymbol{y})}{p(\boldsymbol{x})} = p(\boldsymbol{y}|\boldsymbol{x})\end{aligned}$$

So Boltzman baseded on the codeword lengths and the posterior are indeed one and the same thing!

In our **running example** we use the result for the optimal free energy $\mathcal{F}^*(\boldsymbol{x}) = -\log_2 \sum_{y'} 2^{-l(\mathbf{x},\mathbf{y}')}$ with the length $l(\boldsymbol{x}, 1) = 3$ bits and $l(\boldsymbol{x}, 2) = 4$ bits we get $\mathcal{F}^*(\boldsymbol{x}) = -\log_2(2^{-3} + 2^{-4}) = 2.415$ bits. This is shorter (better) then the naive sampling used above with $q(\mathbf{y}|\boldsymbol{x}) = 0.5$ which gave 2.5 bits.

Alternative ways to write the free energy

$$\begin{aligned}
\mathcal{F}(\boldsymbol{x}) = \mathcal{E}(\boldsymbol{x}) - \mathcal{H}(\boldsymbol{x}) &= \sum_y q(\boldsymbol{y}|\boldsymbol{x}) l(\boldsymbol{x}, \boldsymbol{y}) + \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 q(\boldsymbol{y}|\boldsymbol{x}) \\
&= \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 2^{l(\boldsymbol{x},\boldsymbol{y})} + \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 q(\boldsymbol{y}|\boldsymbol{x}) \\
&= \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 \frac{q(\boldsymbol{y}|\boldsymbol{x})}{2^{-l(\boldsymbol{x},\boldsymbol{y})}} = \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 \frac{q(\boldsymbol{y}|\boldsymbol{x})}{p(\boldsymbol{x}, \boldsymbol{y})} \\
&= \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 \frac{q(\boldsymbol{y}|\boldsymbol{x})}{p(\boldsymbol{y}|\boldsymbol{x}) p(\boldsymbol{x})} = \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 \frac{q(\boldsymbol{y}|\boldsymbol{x})}{p(\boldsymbol{y}|\boldsymbol{x})} - \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 p(\boldsymbol{x}) \ .
\end{aligned}$$

Here we are quickly getting to ELBO! Take the expected free energy

$$\begin{aligned}
\mathbb{E}_{p(\mathbf{x})} \mathcal{F}(\mathbf{x}) &= \sum_x \sum_y p(\mathbf{x}) q(\boldsymbol{y}|\mathbf{x}) \log_2 \frac{q(\boldsymbol{y}|\boldsymbol{x})}{p(\boldsymbol{y}|\mathbf{x})} - \sum_x \sum_y p(\boldsymbol{x}) q(\boldsymbol{y}|\boldsymbol{x}) \log_2 p(\boldsymbol{x}) \\
&= D_{\mathrm{KL}}(q(\mathbf{y}|\mathbf{x}) || p(\mathbf{y}|\mathbf{x})) - \mathbb{E}_{p(\mathbf{x})} \log_2 p(\mathbf{x}) = -ELBO \qquad (2.6)
\end{aligned}$$

How much do we suffer from not using the Boltzman (posterior) $q^*(\boldsymbol{y}|\boldsymbol{x}) = p(\mathbf{y}|\boldsymbol{x})$ but some other distribution $q(\boldsymbol{y}|\boldsymbol{x})$ to sample $\mathbf{y}$? Its the difference in the free energies:

$$\begin{aligned}
\mathcal{F}(\boldsymbol{x}) - \mathcal{F}^*(\boldsymbol{x}) &= \sum_y q(\boldsymbol{y}|\boldsymbol{x}) l(\boldsymbol{x}, \boldsymbol{y}) + \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 q(\boldsymbol{y}|\boldsymbol{x}) + \log_2 p(\boldsymbol{x}) \\
&= \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 \frac{q(\boldsymbol{y}|\boldsymbol{x}) p(\boldsymbol{x})}{2^{-l(\boldsymbol{x},\boldsymbol{y})}} = \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 \frac{q(\boldsymbol{y}|\boldsymbol{x}) p(\boldsymbol{x})}{p(\boldsymbol{x}, \boldsymbol{y})} = \sum_y q(\boldsymbol{y}|\boldsymbol{x}) \log_2 \frac{q(\boldsymbol{y}|\boldsymbol{x})}{p(\boldsymbol{y}|\boldsymbol{x})}
\end{aligned}$$

This is the KL divergence between the sampling distribution $q(\boldsymbol{y}|\boldsymbol{x})$ and the optimal Boltzman or posterior distribution $p(\boldsymbol{y}|\boldsymbol{x}) = q^*(\boldsymbol{y}|\boldsymbol{x})$.

2.2 Bits-back coding algorithm

If $\mathbf{y}$ is not simply two-valued or diadic (powers of two) variable, they propose to use arithmetic coding as the $f : \mathbf{z} \rightarrow \mathbf{y}$ algorithm. I don't really see how that

2.3 Link to maximum likelihood estimation

In the maximum likelihood approach (see section 1) the cost we minimize is the length of codes $c(\boldsymbol{x}) = -\log_2 p_{\boldsymbol{\phi}}(\boldsymbol{x}) = -\log_2 \sum_{\boldsymbol{y}} p_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{y})$.

In section 1 we used $p(\boldsymbol{x}|\boldsymbol{\theta})$ which I here call $p_{\boldsymbol{\phi}}(\boldsymbol{x})$ cause I use $\boldsymbol{\theta}$ already.

For each pair $\boldsymbol{x}, \boldsymbol{y}$ we can use codeword with length $l(\boldsymbol{x}, \boldsymbol{y}) = -\log_2 p_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{y})$ so that the maximum likelihood cost is $c(\boldsymbol{x}) = -\log_2 \sum_{\boldsymbol{y}} 2^{-l(\boldsymbol{x},\boldsymbol{y})} = \mathcal{F}^*(\boldsymbol{x})$.

Maximum-likelihood cost is thus equal to the cost of optimal bits-back coding.

When we do not use the optimal $q(\mathbf{y}|\boldsymbol{x})$ then we get the ELBO equivalence in equation (2.6).

2.4 Experiments

Modelling binary data vectors $\mathbf{x}$ assuming these are produced by a set of hidden binary causes $\mathbf{y}$. The model can produce codeword for each pair of these with length $l(\boldsymbol{x}, \boldsymbol{y})$ (not restricted to be integer length here) so that $\sum_{\boldsymbol{x},\boldsymbol{y}} 2^{-l(\boldsymbol{x},\boldsymbol{y})} \leq 1$.

The above is called *Kraft inequality* and is a standard result in coding theory. My translation is the following: if you always use the optimal length $l^*(\boldsymbol{x}, \boldsymbol{y}) = -\log_2 p(\boldsymbol{x}, \boldsymbol{y})$ then it would be an quality $\sum_{\boldsymbol{x},\boldsymbol{y}} 2^{-l(\boldsymbol{x},\boldsymbol{y})} = \sum_{\boldsymbol{x},\boldsymbol{y}} p(\boldsymbol{x}, \boldsymbol{y}) = 1$. But because you allow for some lengths being inefficient and therefore longer $l(\boldsymbol{x}, \boldsymbol{y}) \geq l^*(\boldsymbol{x}, \boldsymbol{y})$ as if the corresponing $p(\boldsymbol{x}, \boldsymbol{y})$ were somewhat smaller the result is the ≤ 1.

The binary vector $\mathbf{x}$ contains $n_\mathbf{x}$ bits (lenght of the binary string); $\mathbf{y}$ contains $n_\mathbf{y}$ bits. Together this creates $2^{n_\mathbf{x}+n_\mathbf{y}}$ possible messages, each with its own probability mass that needs to be established before encoding.

They use Bayesian network to get these probabilities. Let $\boldsymbol{s} = (\boldsymbol{y}, \boldsymbol{x})$ and use the standard product rule of probability for the vector $p(\boldsymbol{s}) = p(s_1) \prod_{i=2}^{n_\mathbf{x}+n_\mathbf{y}} p(s_i|s_{i-1}, \ldots, s_1)$. They model each probability as a sigmoid $p(s_i|s_{<i}) = \sigma(\theta_i + \sum_{j<i} \theta_{ij} s_j)$, where they will learn the parameters $\boldsymbol{\theta}$.

Furthermore they put some of the $\theta_{ij} = 0$ and use it tu create a hierarchy between the elements of the vector $\mathbf{y}$ essentially transforming it to a two layer binary cause $\mathbf{y} = (\mathbf{c}, \mathbf{z})$ composed of two binary vectors with total length $n_\mathbf{y}$. The lowest layer is modelled as a product distribution $p(\mathbf{c}) = p(\mathbf{c}_1)p(\mathbf{c}_2)p(\mathbf{c}_3) = p(\mathbf{s}_1)p(\mathbf{s}_2)p(\mathbf{s}_3)$ (product goes on depending on how many elements $\mathbf{c}$ has, in figure 2 it has 3).

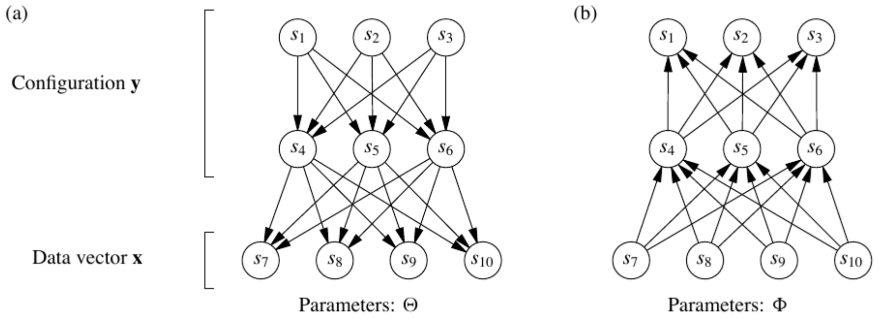
(a)
Configuration **y**
Data vector **x**
s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10}
Parameters: Θ
(b)
s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10}
Parameters: Φ

Figure 2: Bayesian network with two level hidden cuases.

We can now encode any message $\boldsymbol{s} = (\boldsymbol{y}, \boldsymbol{x})$ by encodeing each $\boldsymbol{s}_i$ by its probabiliy given by the Bayes net.

Given $\boldsymbol{x}$ we would preferably want to pick $\boldsymbol{y}$ according to the Boltzman which can also be written as $q^*(\mathbf{y}|\boldsymbol{x}) = p(\boldsymbol{x}, \boldsymbol{y}) / \sum_\mathbf{y} p(\boldsymbol{x}, \boldsymbol{y})$. The sum in the denominator will have $2^{n_\mathbf{y}}$ terms which is clearly not tractable (or not reasonably) for any high dimensional $\mathbf{y}$ so using the Boltzman directly (the posterior) to pick the $\mathbf{y}$ is imposible.

Picking the shortest code is also expensive cause it would have to search among the $2^{n_\mathbf{y}}$ codewords based on $p(\boldsymbol{x}|\boldsymbol{y})$ exhaustively.

Instead of the intractable $q^*(\mathbf{y}|\mathbf{x})$ they will therefore use $q_{\boldsymbol{\phi}}(\mathbf{y}|\mathbf{x})$ with the probabilities given by the reversed network given in figure 2 b. Note that this has reversed the conditional dependencies so that $q(\boldsymbol{s}) = q(s_n) \prod_{i=1}^{n_\mathbf{x}+n_\mathbf{y}-1} q(s_i|s_{i+1}, \ldots, s_n)$ allowing also for the zeroing of paramters and hence the two-lavel cuses hieararchy. The probabilities are again sigmoids, this time with $\boldsymbol{\phi}$ learned parameters.

The bits-back algo then works like this: The *sender* first computes $q(s_6|s_7,\ldots,s_{10})$ (using part b of figure 2) and uses auxiliary data together with arithmetic decoder to obtain s_6. It then computes $q(s_5|s_6,\ldots,s_{10})$ and uses auxiliary data and arithmetic decoder to get s_5 and so on goes back bit by bit until it gets $s_1,\ldots,s_6$. Once done, it encodes bit by bit $s_1,\ldots,s_{10}$ using arithmetic encoder with probabilities from part a of figure 2 and transmits the codeword. The *receiver* decodes from the codeword the message $s_1,\ldots,s_{10}$ and then proceeds bit by bit to recover the auxiliary data again bit by bit.

They experiment on $\mathbf{x} \in 0,1^{8\times 8}$ images of digits so that $n_{\mathbf{x}} = 64$ and they fix $n_{\mathbf{y}} = 60$ in 3 layer hierarchy.

They assume the auxiliary data is useful but do not actually specify what they really are. They acknowledge it is not quite clear how to use part of the primary data as auxiliary but they are looking into it.

3 Townsend: Tutorial on rANS

Paper: James Townsend. *A Tutorial on the Range Variant of Asymmetric Numeral Systems.* 2020. URL: `http://arxiv.org/abs/2001.09186` (visited on 06/02/2021)

Notes taken: 9/6/2021

3.1 Intro

All log should be read as $\log_2$ here!

The difference between the range variant of *asymetric numeral systems* (*rANS*) and *arithmetic coding* (*AC*) is the order in which they are decoded: in rANS compression is LIFO (stack-like), in AC it is FIFO (queue-like).

There are two basic functions in ANS:

- `push` : $(m, x) \to m'$
- `pop` : $m' \to (m, x)$,

where m is a compressed message and x is an uncompressed symbol. These are inverses of each other so that it holds $\texttt{push}(\texttt{pop}(m)) = m$ and $\texttt{pop}(\texttt{push}(m, x)) = (m, x)$.

An ensamble X with precision r is a triple $(x, \mathcal{A}_X, \mathcal{P}_X)$ where the outocme x takes a value out of the set of possible values $\mathcal{A}_X = \{a_1, \dots, a_I\}$ and $\mathcal{P}_X = \{p_1, \dots, p_I\}$ are the integer valued probability weights such that $\sum_i^I p_i = 2^r$ and the probability $P(x = a_i) = p_i/2^r$. So the probabilities are quantized to some precision r (with $r = 23$ we get about the 32-bit floating precision so not a limiting assumption).

The *Shannon information content* of an outcome x is $h(x) = -\log P(x)$.

For a sequence of ensambles $X_1, \dots, X_N$ we seak algo that can encode the sequence of outcomes $x_1, \dots, x_N$ in a binary message with length close $h(x_1, \dots, x_N) = -\log P(x_1, \dots, x_N)$ (in expectation).

By the standard probability rules we have

$$h(x_1, \dots, x_N) = -\log P(x_1, \dots, x_N) = -\sum_n \log P(x_n | x_1, \dots, x_{n-1}) = \sum_n h(x_n | x_1, \dots, x_{n-1}) \ .$$

3.2 ANS

We structure the message $m = (s, t)$, where s in unsigned integer with precision r_s, $s \in \{0, 1, \dots, 2^{r_s} - 1\}$ so that it can be expressed as binary number with r_s bits. Next t is a stack of unsigned integers with precision $r_t < r_s$ with its own `stack_push` and `stack_pop` operations. We also impose a constraint $s \geq 2^{r_s - r_t}$. We obtain message m by contatenating s and the elements of t.

The **length** of m is $l(m) := r_s + r_t|t|$, where $|t|$ is the number of elements in the stack t. The **effective length** of m is $l^*(m) := \log s + r_t|t|$. Since $2^{r_s - r_t} \leq s < 2^{r_s}$ we have $r_s - r_t \leq \log s < r_s$ and hence

$$l(m) - r_t \leq l^*(m) < l(m) \ . \tag{3.1}$$

Intuitively $l^*(m)$ is the precise measure of the size of m while $l(m)$ is the integer-valued length - the length of the binary code to be transmitted.

3.2.1 Decoder pop

The **decoder** decodes the binary message m by performing a series of N `pop` operations starting from m_0:

$$(m_n, x_n) = \texttt{pop}(m_{n-1}), \quad n = 1, \ldots, N \tag{3.2}$$

Each `pop` uses the probability $P(X_n|X_1, \ldots, X_{n-1})$

First split m to (s, t), then extract symbol x from s via bijective function $d : \mathbb{N} \to \mathbb{N}\times\mathcal{A},\ s \to (s', x)$. d is such that if $s \geq 2^{r_s - r_t}$

$$\log s - \log s' \leq h(x) + \epsilon, \qquad \epsilon = \log \frac{1}{1 - 2^{-(r_s - r_t - r)}} \tag{3.3}$$

For small $2^{-(r_s-r_t-r)}$ we can approximate $\epsilon \approx \frac{2^{-(r_s-r_t-r)}}{\log 2}$. They use $r_s = 64, r_t = 32, r = 16$ which gives $\epsilon = \log \frac{1}{1-2^{-16}} = 2.2 \times 10^{-5}$ so very small.

We then check if s' is smaller then $2^{r_s - r_t}$, if yes we `stack_pop` integers from t to lower order bits of s'.

Data: Some input data
these inputs can be displayed on several lines and oneinput can be wider than line's width.

References

[1] Brendan J. Frey and Geoffrey E. Hinton. “Efficient Stochastic Source Coding and an Application to a Bayesian Network Source Model”. In: *Computer Journal* 40.2/3 (1997), pp. 157–165 (cit. on p. 3).

[2] James Townsend. *A Tutorial on the Range Variant of Asymmetric Numeral Systems.* 2020. URL: `http://arxiv.org/abs/2001.09186` (visited on 06/02/2021) (cit. on p. 9).

Index